竞学实训-CAN Security 实验报告

lwenfg

2025 年 7 月 21 日

目录

1 实验目的

- 理解汽车 CAN 总线通信协议的安全机制与脆弱性
- 掌握基于 ICSim 的 CAN 网络模拟环境搭建
- 实操常见 CAN 攻击技术并分析防御方案

2 模拟 CAN 信号通信

2.1 虚拟 CAN 接口创建

创建模拟一个虚拟 CAN 接口，分别输入下面的指令：

```
sudo modprobe vcan

sudo ip link add dev vcan0 type vcan

sudo ip link set up vcan0
```

后续若有需要可以使用以下命令对 vcan0 接口进行删除：

```
sudo ip link delete vcan0
```

2.2 模拟信号通信

请使用 can-utils 工具包里的不同工具（如 candump、cansend、cangen 等）实现 CAN 报文的发送和嗅探。(1) 使用 candump 实时监听和显示 CAN 总线上的原始数据帧，在终端中输入下面的指令

```
candump vcan0
```

(2) 新开一个终端，使用 candsend 发送特定的数据帧，输入下面指令后可在 candump 终端里看到所发送的数据。

```
cansend vcan0 123#AABBCCDD
```

(3) 新开一个终端，使用 cangen 随机发送一些数据帧，同样可在 candump 终端里看到所发送的数据。

```
cangen vcan0
```

3 基于 ICSim 的基础实践

3.1 ICSim 模拟环境搭建

官方仓库如下 https://github.com/zombieCraig/ICSim
安装并编译过程的指令如下：

```
sudo apt-get update
sudo apt-get install libsdl2-dev libsdl2-image-dev can-utils

git clone https://github.com/zombieCraig/ICSim.git

cd ICSim
make
```

以当前进行实验时的年月日组合为种子（例如：20250713）正确启动，可以通过键盘或者手柄完成对仪表盘功能的控制（转向灯控制、加速控制、车门控制）。

在 ICSim 文件夹下进入终端，首先输入下面的指令启动仪表盘（一定要先启动仪表盘再启动控制器）

```
./icsim -s 20250713 vcan0
```

再输入以下指令启动控制器

```
./controls -s 20250713 vcan0
```

在仪表盘和控制器显示出来后，要想对汽车进行控制，首先用光标点击控制器（否则无感应），按住键盘左右键可以控制转向灯，按住左或右 ctrl+A、B、X、Y 可以控制车门的解锁和关闭，按住上下键可以控制加速和减速。在需要退出仪表盘和显示器时可以在命令台输入 ctrl+c.

3.2 Can-utils 使用

使用 can-utils 工具包里的相关工具完成对 CAN 数据报文的监听和转储，通过肉眼观察或代码分析，找到与转向灯控制、车门控制、加速控制相关的 CAN 数据帧。

这一部分比较繁琐，需要使用肉眼观察相关的数据帧。具体操作为使用 can-utils 工具包中的 candump 指令和 cansniffer。指令进行观察。比较推荐使用 cansniffer。

在新命令台输入下面的指令，在操纵车辆按键时观察是否有新的数据帧出现，若有新的数据帧出现则为该操作相关的数据帧。

```
cansniffer -c vcan0
```

以寻找转向灯数据帧为例，通过对比发现在控制转向灯后新出现的数据帧的 ID 为 2A3，所以可以大体判断出 2A3 即为转向灯的数据帧，控制字段中第二个字节为转向灯的控制，例如，00010000 代表左转向灯，00020000 代表右转向灯，00000000 代表转向灯关闭，对比结果如下图所示。

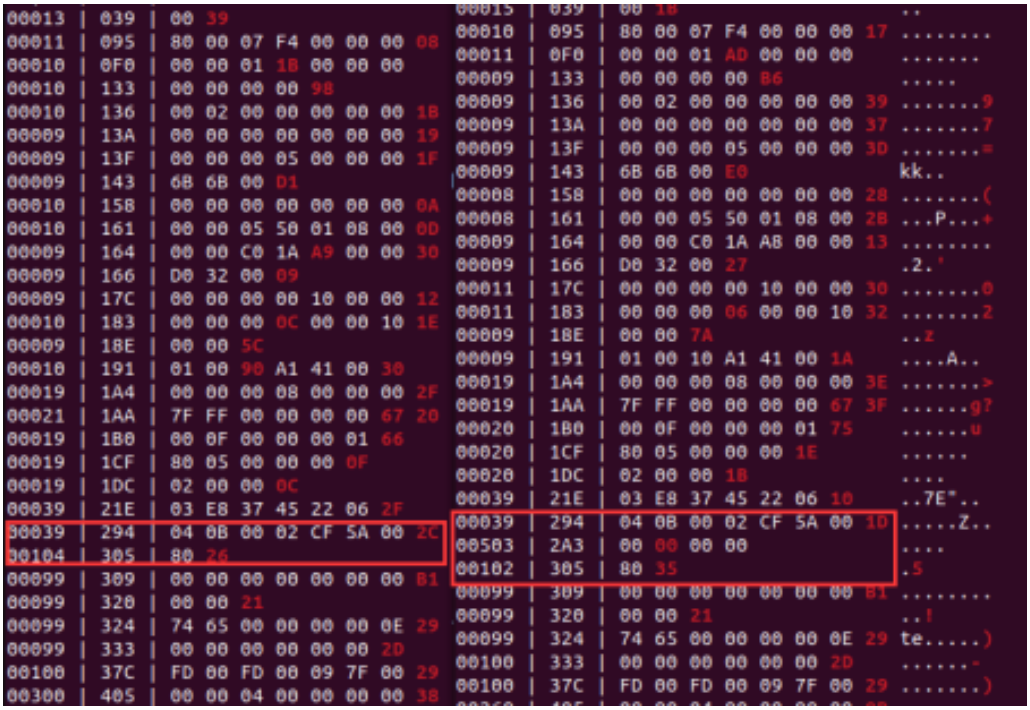

图 1: 对比界面

门控，加速控制相关数据帧可以使用同样的方法找到。但是加速数据帧有些特殊，开关门和打开转向灯都是“离散”的信号，但在加速过程中，速度不断增大，所以对应的控制字段的字节也在不断增大，信号是“连续的”。我们需要找到控制字段中不断增加的字节，其所在的数据帧即为与速度相关的数据帧。

4 SavvyCAN 使用

4.1 SavvyCAN 的安装

在安装过程中输入以下指令，首先克隆 SavvyCAN 仓库，下载必要的依赖包，并进行编译。在安装过程中，若遇到的克隆时连接失败，解决方案为科学上网或者在 Windows 中下载好再拖入虚拟机中。若使用后者的方法，需要在虚拟机中设置环境变量路径、增加权限后才能使用指令./SavvyCAN

```
git clone https://github.com/collin80/SavvyCAN.git

sudo apt install libqt5serialbus5-dev libqt5serialport5-dev
sudo apt install qtdeclarative5-dev qttools5-dev

cd SavvyCAN
~/Qt5.14.2/5.14.2/gcc_64/bin/qmake

make
```

在编译好后，打开命令台，输入以下指令即可启动 SavvyCAN：

```
./SavvyCAN
```

4.2 SavvyCAN 使用

使用 SavvyCAN，监听你创建的 CAN 接口，探索 SavvyCAN 的不同功能，并通过其 FUZZ 功能筛选灯控、门控、加速的关键数据帧。

（1）首先在 Ubuntu 上创建 vcan0 接口，并打开相应的仪表盘和显示器，然后在 SavvyCAN 中创建 CAN 接口，连接到 vcan0 上，如图所示：

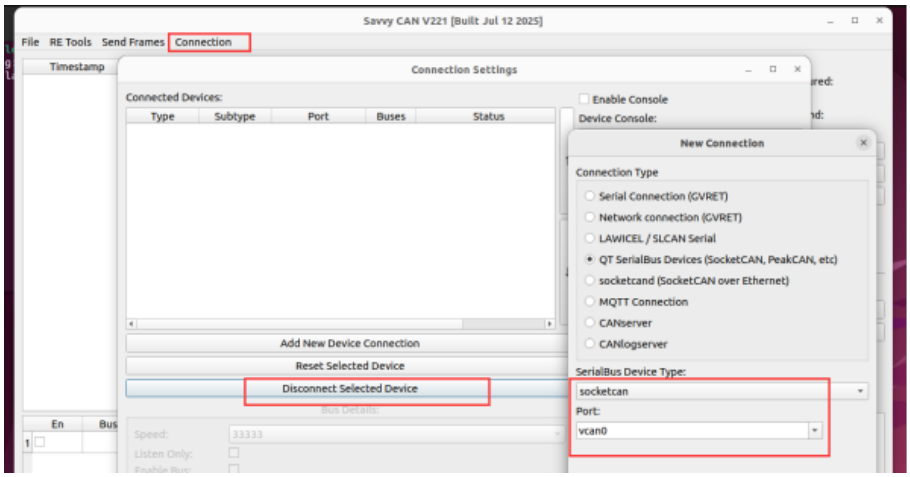

图 2: 创建步骤

（2）接着进行监控，首先使用 Sniffer 功能对关键数据帧进行探查，获取灯控、门控和加速的有关数据帧的 ID，方法和 can-utils 工具包的相似，用肉眼观察新增的 ID 或者加速时数据字段不断增长的 ID。最终得出的结果如下：灯控数据帧 ID 为 0x2A3，门控数据帧的 ID 为 0x02C，加速数据帧 ID 为 0x0F0，寻找转向灯数据帧过程如图所示：

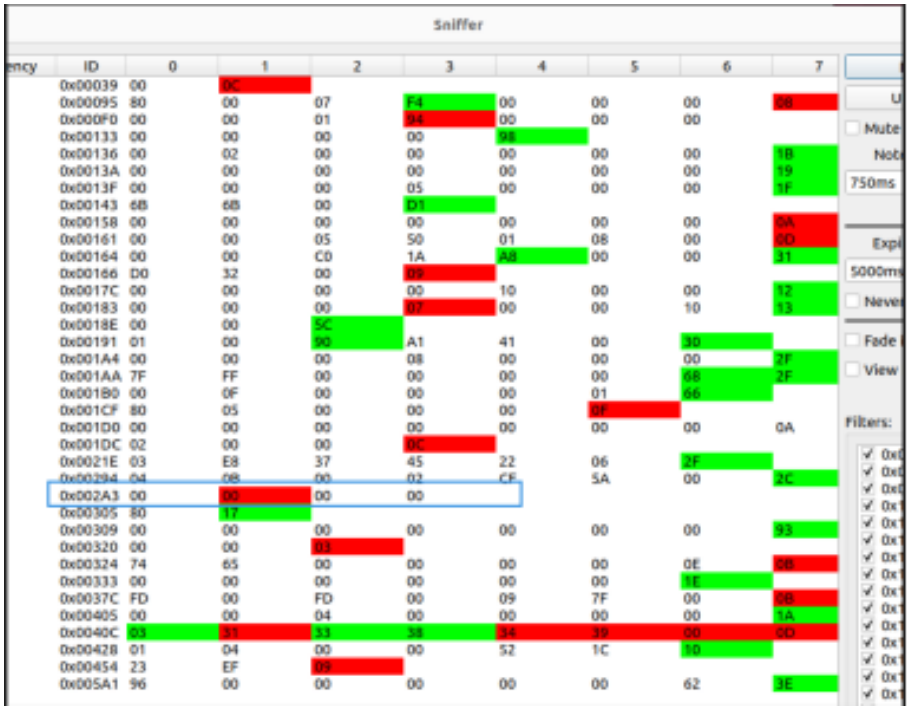

图 3: 使用 SavvyCAN 寻找转向灯数据帧

（3）接着使用 FUZZ 功能查找对应 ID 的数据帧，首先筛选出对应的数据帧 ID，对该数据帧中对每一个字节和每一位都进行模糊测试，找出是哪个字节对操作有关。一个简单的例子如图所示，筛选出与灯控相关的数据帧 ID 为 0x2A3，对该数据帧每一个字节进行模糊测试，看仪表盘中的转向灯是否有变化。若有变化则该字节即为用于控制转向灯的字节。再对字节中每一位进行模糊测试，判断具体是哪一位。

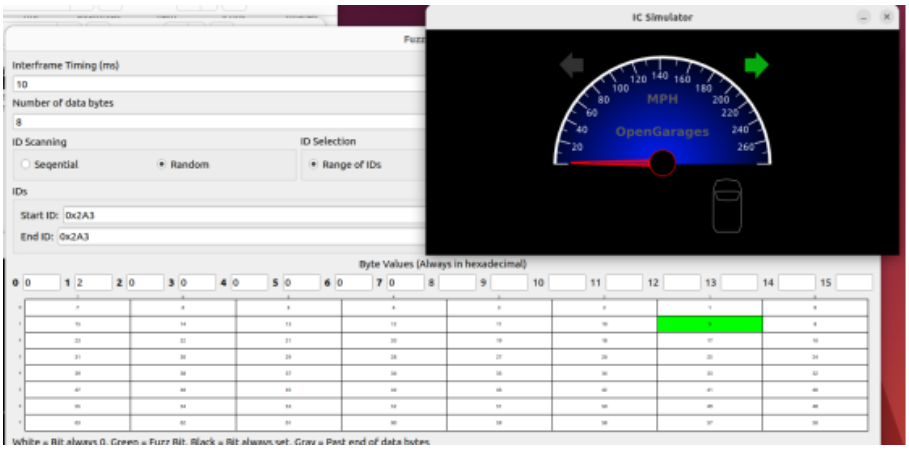

图 4: 使用 FUZZ 寻找转向灯相关字节

4.3 关键帧总结

列出你所得到的与灯控、门控、加速有关的关键帧详细信息（报文 ID、数据长度、关键控制字段在哪一个或多个字节）。

1. 灯控：报文 ID 为 0x2A3，数据长度为四个字节，关键控制字段在索引为 0 的字节。在该字节中最后一位控制左转向灯，倒数第二位控制右转向灯。

2. 门控：报文 ID 为 0x02C，数据长度为六个字节，关键控制字段在索引为 3 的字节。在该字节中，最后四位分别控制着四个门。

3. 加速：报文 ID 为 0x0F0，数据长度为七个字节，关键控制字段在索引为 2 的字节。在该字节中，所有位都参与了对加速的控制。

5 进阶实验

5.1 关键帧分析

以你的个人学号为种子，在难度 2 下重新启动 ICSim 模拟器 3，使用 SavvyCAN，找到当前模拟条件下与灯控、门控、加速有关的关键帧详细信息。

首先在打开两个命令台，分别输入下面的指令（注意顺序和难度设置），以学号为种子，在难度 2 的情况下打开 ICSim 模拟器。

```
./icsim -s 20230046xxxx vcan0

./controls -s 20230046xxxx -l 2 vcan0
```

接着打开 SavvyCAN，并进行端口的连接，跟上述方法相同打开 SavvyCAN 的 Sniffer 功能和 FUZZ 功能分别查找对应的 ID 和相关的数据字段的字节，此处不再一一赘述和展示，最终得到的结果为：

1. 灯控：报文 ID 为 0x5D3，关键控制字段在索引为 3 的字节。在该字节中最后一位控制左转向灯，倒数第二位控制右转向灯。
2. 门控：报文 ID 为 0x74E，关键控制字段在索引为 2 的字节。在该字节中，最后四位分别控制着四个门。
3. 加速：报文 ID 为 0x720，关键控制字段在索引为 1 的字节。在该字节中，所有位都参与了对加速的控制。

5.2 攻击模拟

5.2.1 重放攻击

重放你之前分析所得到的关键单帧以及在某一段操作时间内用 candump 捕获到的所有数据帧日志，观察是否能够控制某些关键功能（灯控、加速、门控）。

以转向灯控制为例，首先根据上述实验所获得的 ID 和相应的字节，重放灯控的数据帧，输入以下指令操纵两个灯同时亮起：

```
cansend vcan0 5D3#0000000300000000
```

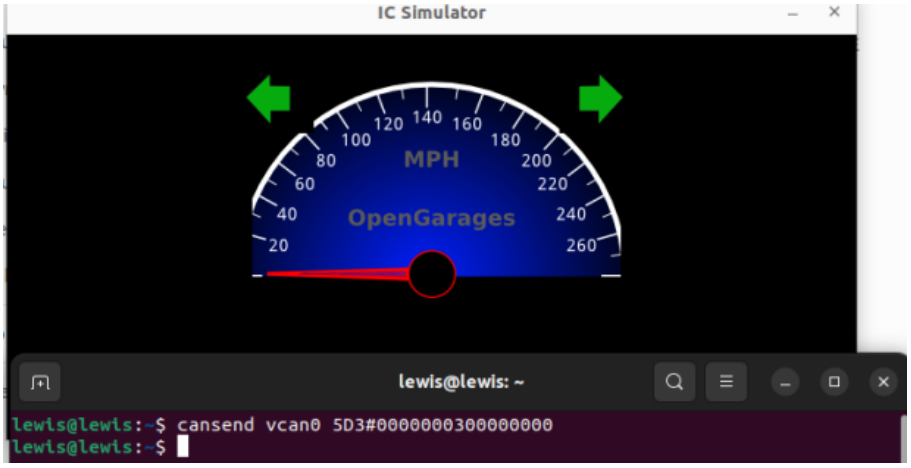

图 5: 重放数据帧使得两个转向灯打开

也可以使用 candump 捕获数据帧日志，观察上述指令是否能够控制灯控的功能。

```
candump vcan0 | grep "5D3"
```

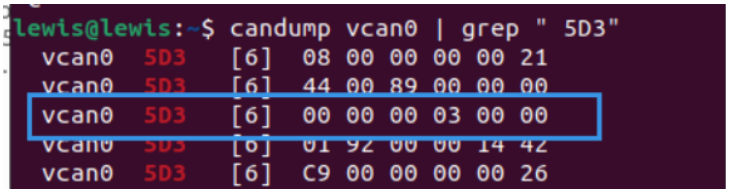

图 6: 查看日志找到重放数据帧

5.2.2 DoS 攻击

用你自己的方式，对灯控、加速、门控服务实施 DoS 攻击，具体表现分别为

1. 左转向灯无法正常亮起、右转向灯可正常控制。
2. 仪表盘失灵，加速时无法正确显示当前速度。
3. 左前门无法正常关闭，其余车门可正常控制。

完整代码见 DOS.c. 由于左转向灯控制字段位于 ID0x5D3 的索引为 3 的字节的第 0 位，所以不断监听并截获该数据帧，当这个数据帧第 3 字节第 0 位为 1 时将其设置为 0，而保持其他位不变（保持右转向灯正常控制），然后再重新发送。代码如下：

```
//转向灯的控制
if (frame.can_id == 0x5D3 && (frame.data[3] & 0x01) != 0)
{
    frame.data[3] = frame.data[3] & 0xFE; // 关闭左转向灯
    write(s, &frame, sizeof(frame));
}
```

同理，对于左前门无法正常关闭而其他车门可以正常控制的情况，只需要在监听 ID 为 0x74E 的数据帧里索引为 2 的字节，将第 0 位设置为 0 即可（1 代表关门，0 代表开门），其他位同样不变，代码如下：

```
//车门的控制
if (frame.can_id == 0x74E && (frame.data[2] & 0x01) != 0)
{
    frame.data[2] = frame.data[2] & 0xFE; // 保持打开左前车门
    write(s, &frame, sizeof(frame));
}
```

因为仪表盘控制字段位于 ID0x720 处的索引为 1 的字节的所有位，所以只需要将该字节置为 0 即可实现仪表盘无法正确显示速度，代码如下：

```
if (frame.can_id == 0x720)
{
    frame.data[1] = 0x00;
    write(s, &frame, sizeof(frame));
}
```

6 实验思考

1.CAN 总线通信与其他网络通信协议相比，有哪些脆弱性？这导致了 CAN 易受哪些攻击？

CAN 总线的脆弱性源于其无认证、无加密、广播式通信的设计特性，这导致其极易遭受三类攻击：（1）注入攻击，攻击者可以任意插入恶意数据帧（如伪造加速或门控指令）；（2）DoS 攻击，通过高频错误帧使总线拥塞瘫痪；（3）ECU 逆向劫持、重放攻击，利用总线无加密特性监听通信规

律（如通过 candump 捕获转向灯 ID），进而重放或模糊测试控制关键节点。

2. 对于 CAN 总线而言，除一般的欺骗、注入、重放、DoS 类攻击之外，还有哪些可能存在的攻击思路？

CAN 总线依靠一对导线上的电压差来传递信号。逻辑 0 对应较大的差分电压，逻辑 1 对应接近 0 的差分电压。攻击者可以选择性地强行拉高或拉低其中一根导线的电压, 从而破坏 CAN 总线上正常的差分电压平衡，直接在物理层面引发错误状态，干扰所有通信。

3. 有哪些防御手段可以增强车内 CAN 总线通信的安全性？

（1）实施 CAN 总线入侵检测系统，通过实时监控总线流量，利用帧频率分析、ID 序列校验和熵值检测等技术识别恶意注入或篡改行为。

（2）采用网络分区隔离，通过网关防火墙将域隔离成独立子网，设置严格的 ID 过滤规则，结合速率限制，防止攻击者入侵整个系统。